```matlab
%% Machine Learning
% Lab 7: Neural Network Training
% —— Handwritten Digits ——
%{
For this exercise, you will teach a neural network to
recognize handwritten digits (from 0 to 9). Automated handwritten digit
recognition is widely used today — from recognizing zip codes (postal codes)
on mail envelopes to recognizing amounts written on bank checks.
%}

%% Initialization
clear ; close all; clc

%% Setup the parameters you will use for this exercise
input_layer_size  = 400;  % 20x20 Input Images of Digits
hidden_layer_size = 25;   % 25 hidden units
num_labels = 10;          % 10 labels, from 1 to 10
                          % (note that we have mapped "0" to label 10)

%% =========== Part 1: Loading and Visualizing Data =============
%  We start the exercise by first loading and visualizing the dataset.
%  You will be working with a dataset that contains handwritten digits.
%

% Load Training Data
fprintf('Loading and Visualizing Data ...\n')

load('ex4data1.mat');
m = size(X, 1);

% Randomly select 100 data points to display
sel = randperm(size(X, 1));
sel = sel(1:100);

displayData(X(sel, :));

fprintf('Program paused. Press enter to continue.\n');
pause;



%% ================ Part 2: Loading Parameters ================
% In this part of the exercise, we load some pre—initialized
% neural network parameters.

fprintf('\nLoading Saved Neural Network Parameters ...\n')

% Load the weights into variables Theta1 and Theta2
load('ex4weights.mat');
w1 = Theta1;
w2 = Theta2;

% Unroll parameters
nn_params = [w1(:) ; w2(:)];

%% ================ Part 3: Compute Cost (Feedforward) ================
```

```
%  To the neural network, you should first start by implementing the
%  feedforward part of the neural network that returns the cost only. You
%  should complete the code in nnCostFunction.m to return cost. After
%  implementing the feedforward to compute the cost, you can verify that
%  your implementation is correct by verifying that you get the same cost
%  as us for the fixed debugging parameters.
%
%  We suggest implementing the feedforward cost *without* regularization
%  first so that it will be easier for you to debug. Later, in part 4, you
%  will get to implement the regularized cost.
%
fprintf('\nFeedforward Using Neural Network ...\n')

% Weight regularization parameter (we set this to 0 here).
lambda = 0;

C = nnCostFunction(nn_params, input_layer_size, hidden_layer_size, ...
                   num_labels, X, y, lambda);

fprintf(['Cost at parameters (loaded from ex4weights): %f '...
         '\n(this value should be about 0.287629)\n'], C);

fprintf('\nProgram paused. Press enter to continue.\n');
pause;

%% =============== Part 4: Implement Regularization ===============
%  Once your cost function implementation is correct, you should now
%  continue to implement the regularization with the cost.
%


fprintf('\nChecking Cost Function (w/ Regularization) ... \n')

% Weight regularization parameter (we set this to 1 here).
lambda = 1;

C = nnCostFunction(nn_params, input_layer_size, hidden_layer_size, ...
                   num_labels, X, y, lambda);

fprintf(['Cost at parameters (loaded from ex4weights): %f '...
         '\n(this value should be about 0.383770)\n'], C);

fprintf('Program paused. Press enter to continue.\n');
pause;


%% ================ Part 5: Sigmoid Gradient  ================
%  Before you start implementing the neural network, you will first
%  implement the gradient for the sigmoid function. You should complete the
%  code in the sigmoidGradient.m file.
%

fprintf('\nEvaluating sigmoid gradient...\n')

g = sigmoidGradient([-1 -0.5 0 0.5 1]);
fprintf('Sigmoid gradient evaluated at [-1 -0.5 0 0.5 1]:\n  ');
fprintf('%f ', g);
fprintf('\n\n');
```

```
fprintf('Program paused. Press enter to continue.\n');
pause;


%% ================ Part 6: Initializing Pameters ================
%  In this part of the exercise, you will be starting to implment a two
%  layer neural network that classifies digits. You will start by
%  implementing a function to initialize the weights of the neural network
%  (randInitializeWeights.m)

fprintf('\nInitializing Neural Network Parameters ...\n')

init_w1 = randInitializeWeights(input_layer_size, hidden_layer_size);
init_w2 = randInitializeWeights(hidden_layer_size, num_labels);

% Unroll parameters
initial_nn_params = [init_w1(:) ; init_w2(:)];

%% =============== Part 7: Implement Regularization ===============
%  Once your backpropagation implementation is correct, you should now
%  continue to implement the regularization with the cost and gradient.
%

fprintf('\nChecking Backpropagation (w/ Regularization) ... \n')

lambda = 3;

% Also output the costFunction debugging values
debug_C  = nnCostFunction(nn_params, input_layer_size, ...
                          hidden_layer_size, num_labels, X, y, lambda);

fprintf(['\n\nCost at (fixed) debugging parameters (w/ lambda = %f): %f ' ...
         '\n(for lambda = 3, this value should be about 0.576051)\n\n'], lambda,
             debug_C);

fprintf('Program paused. Press enter to continue.\n');
pause;


%% =================== Part 9: Training NN ===================
%  You have now implemented all the code necessary to train a neural
%  network. To train your neural network, we will now use "fmincg", which
%  is a function which works similarly to "fminunc". Recall that these
%  advanced optimizers are able to train our cost functions efficiently as
%  long as we provide them with the gradient computations.
%
fprintf('\nTraining Neural Network... \n')

%  After you have completed the assignment, change the MaxIter to a larger
%  value to see how more training helps.
options = optimset('MaxIter', 50);

%  You should also try different values of lambda
lambda = 1;

% Create "short hand" for the cost function to be minimized
```

```
costFunction = @(p) nnCostFunction(p, ...
                                   input_layer_size, ...
                                   hidden_layer_size, ...
                                   num_labels, X, y, lambda);

% Now, costFunction is a function that takes in only one argument (the
% neural network parameters)
[nn_params, cost] = fmincg(costFunction, initial_nn_params, options);

% Obtain w1 and w2 back from nn_params
w1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size + 1)), ...
                 hidden_layer_size, (input_layer_size + 1));

w2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size + 1))):end), ...
                 num_labels, (hidden_layer_size + 1));

fprintf('Program paused. Press enter to continue.\n');
pause;


%% ================= Part 9: Visualize Weights =================
%  You can now "visualize" what the neural network is learning by
%  displaying the hidden units to see what features they are capturing in
%  the data.

fprintf('\nVisualizing Neural Network... \n')

displayData(w1(:, 2:end));

fprintf('\nProgram paused. Press enter to continue.\n');
pause;

%% ================= Part 10: Implement Predict =================
%  After training the neural network, we would like to use it to predict
%  the labels. You will now implement the "predict" function to use the
%  neural network to predict the labels of the training set. This lets
%  you compute the training set accuracy.

pred = predict(w1, w2, X);

fprintf('\nTraining Set Accuracy: %f\n', mean(double(pred == y)) * 100);
```

nnCostFunction.m

```
function [C grad] = nnCostFunction(nn_params, ...
                                   input_layer_size, ...
                                   hidden_layer_size, ...
                                   num_labels, ...
                                   X, y, lambda)
%NNCOSTFUNCTION Implements the neural network cost function for a two layer
%neural network which performs classification

% Reshape nn_params back into the parameters weight1 and weight2, the weight
    matrices
% for our 2 layer neural network
w1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size + 1)), ...
                 hidden_layer_size, (input_layer_size + 1));

w2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size + 1))):end), ...
                 num_labels, (hidden_layer_size + 1));

% Setup some useful variables
m = size(X, 1);

% You need to return the following variables correctly
C = 0;
w1_grad = zeros(size(w1));
w2_grad = zeros(size(w2));

A1 = [ones(size(X,1),1) X];
S2 = A1*w1';
A2 = sigmoid(S2);
A2_0 = [ones(size(A2,1),1) A2];
S3 = A2_0*w2';
A3 = sigmoid(S3);

HX = A3;

% recode y to Y // TRICKY => ONE HOT ENCODING
I = eye(num_labels);
Y = zeros(m, num_labels);

for i=1:m
  Y(i, :)= I(y(i), :);
end

penalty = (lambda /(2*m))*(sum(sum(w1(:, 2:end).^2, 2))+sum(sum(w2(:, 2:end).^2, 2)
    ));

C = (1/m) * sum(sum((-Y).*log(HX)-((1-Y).*log(1-HX))))+penalty;

% calculate sigmas //DELTAS
sigma3 = (A3-Y);
sigma2 = (sigma3*w2).*sigmoidGradient([ones(size(S2, 1), 1) S2]);
sigma2 = sigma2(:, 2:end);

% accumulate gradients
delta_1 = (sigma2'*A1);
delta_2 = (sigma3'*A2_0);

```

```
% calculate regularized gradient
p1 = (lambda/m)*[zeros(size(w1, 1), 1) w1(:, 2:end)];
p2 = (lambda/m)*[zeros(size(w2, 1), 1) w2(:, 2:end)];
w1_grad = delta_1./m + p1;
w2_grad = delta_2./m + p2;

% Unroll gradients
grad = [w1_grad(:) ; w2_grad(:)];

end
```

sigmoidGradient.m

```
function g = sigmoidGradient(z)
%SIGMOIDGRADIENT returns the gradient of the sigmoid function
%evaluated at z

g = zeros(size(z));
g = sigmoid(z).*(1-sigmoid(z));

end
```

randInitializeWeights.m

```
function W = randInitializeWeights(L_in, L_out)
%RANDINITIALIZEWEIGHTS Randomly initialize the weights of a layer with L_in
%incoming connections and L_out outgoing connections
%   Note that W should be set to a matrix of size(L_out, 1 + L_in) as
%   the first column of W handles the "bias" terms

W = zeros(L_out, 1 + L_in);
% Initialize W randomly so that we break the symmetry while
% training the neural network.

epsilon_init = 0.12;
W = rand(L_out, 1 + L_in) * 2 * epsilon_init - epsilon_init;

end
```

predict.m

```
function p = predict(w1, w2, X)
%PREDICT Predict the label of an input given a trained neural network
%   p = PREDICT(w1, w2, X) outputs the predicted label of X given the
%   trained weights of a neural network (w1, w2)

% Useful values
m = size(X, 1);
num_labels = size(w2, 1);

% You need to return the following variables correctly
p = zeros(size(X, 1), 1);

h1 = sigmoid([ones(m, 1) X] * w1');
h2 = sigmoid([ones(m, 1) h1] * w2');
[dummy, p] = max(h2, [], 2);

end
```